



BRADLEY
University

PANDUIT®

Wireless Client Location Determination System

Team Members: Nicholas Pratt, Michael Stanczyk, Nathan Ruetten

Project Advisor: Dr. Aleksander Malinowski

Client: Dr. Ronald Nordin, Panduit Inc.

Department of Electrical and Computer Engineering

March 28, 2017

ABSTRACT

Many organizations, including Panduit, currently utilize various types of wireless sensors for many applications such as temperature and pressure measurements throughout an environment. It can be advantageous to know the physical location of these wireless sensors. Conventional location determination systems, such as the Global Positioning System (GPS), cannot be used in certain environments and a different solution is needed. In some cases, sensor locations can be manually assigned and fixed. However, for deploying a great number of sensors, it would be easier to have this location be determined automatically and allow the sensors to be moved through a closed environment. Our research focuses on designing a location determination system using IEEE 802.11 Wi-Fi to determine the location of a sensor with an accuracy of 1 meter or less in two-dimensional or three-dimensional space.

Table of Contents

INTRODUCTION	4
PROBLEM STATEMENT	4
COMPARISON OF WIRELESS LOCATION DETERMINATION METHODS	5
Criteria of Comparison	5
Comparison of Possible Solutions	6
EVALUATION OF THE SUBPOS SYSTEM	9
DETERMINATION OF A CUSTOM DISTANCE DETERMINATION EQUATION	10
TRILATERATION / LOCALIZATION	14
DEVELOPMENT OF A SERVER	17
CONCLUSIONS AND FUTURE WORK	19
APPENDIX A - PARTS LIST	23
APPENDIX B - SERVER SOURCE CODE	24
APPENDIX C - CLIENT SOURCE CODE	35

I. INTRODUCTION

Wireless sensors, sensors that communicate their data via some sort of wireless protocol, are used by countless organizations for various reasons such as measurements of room temperature, gear speed, or valve pressures. The advantage of using these sensors over their wired counterparts is the ability to gather their information remotely. As the Internet of Things (IoT) movement continues to grow, the volume of wireless devices is expected to increase dramatically and wireless sensors represent one example of this.

Knowing the physical locations of these sensors is advantageous as the sensors may need servicing or could indicate a problem with their surroundings. The physical locations of sensors could be manually assigned, but this makes it more difficult for sensors to be relocated, added, or removed, and as large numbers of sensors are deployed, it becomes more complicated to keep track of hundreds or thousands of locations. Sensors being utilized outdoors could be located through the use of the Global Positioning System (GPS). Since GPS signals are greatly attenuated inside buildings, this is not a practical solution for indoor wireless sensors or where GPS is not available.

While there are many types of radio frequency (RF) protocols that could be used for location determination, it is expected that Wi-Fi (based on IEEE 802.11 specifications) will be the most widely used for reasons to be discussed.

II. PROBLEM STATEMENT

Dr. Ronald Nordin of Panduit, Inc. approached Bradley University with a proposal for researching and developing a prototype for a wireless location determination system. The project would research and compare several methods of wireless location determination, propose a solution of the best method, and prototype the method to act as a proof-of-concept.

Five key properties of an ideal, robust wireless location determination system were defined:

1. The system should support sensors that can simply be placed and turned on with no additional setup required.
2. The system needs to work with any wireless sensor. The infrastructure itself must handle all location determination calculations and measurements.
3. The system must conform to all governmental regulations for wireless protocols.
4. The system should continuously monitor the environment for new, missing, or changed sensors.
5. The system should be accurate to within an ideal 1 meter³.

While all five points are important, wireless location determination systems currently exist but with a less-than-desired accuracy. As the number of sensors grows, a system which can determine location to an accuracy of within 1 meter in 3-space would be ideal.

III. COMPARISON OF WIRELESS LOCATION DETERMINATION METHODS

A. Criteria of Comparison

According to researcher Junjie Liu [1], a wireless based indoor location determination system consists of “beacon stations that emit the wireless signal and the user devices that receive the signal or vice versa”. In the case of the proposed project, the logic for computing location would reside with the beacon stations as the wireless sensors must remain as unchanged as possible.

While there are several other wireless technologies that could be used for location determination including ZigBee, Frequency Modulation (FM), Bluetooth, and RFID, Wi-Fi is the most fruitful in terms of research. FM has the greatest range of all the above technologies, but would be too granular and not provide enough signal data to determine location to the accuracy desired. Bluetooth and RFID are short-range technologies and would require a significant deployment cost to provide enough beacons. ZigBee has similar specifications to Wi-Fi, but requires a dedicated and complex mesh infrastructure. While several of these technologies could be combined, the complexity of such a network would likely exceed the scope of this project.

Liu provides several criteria for evaluating the performance of a wireless localization system. These include:

1. Accuracy - How close to the true physical location does the system calculate?
2. Precision - How consistent is the output of the location calculation method?
3. Coverage - How much area is covered by a single beacon?
4. Update Interval - How often does the system update the locations of the sensors?
5. Computational Cost - What is the expense in terms of power in order to compute locations?
6. Infrastructure - What is the expense for deploying and maintaining the infrastructure itself?
7. Offline Computing - What is the expense in terms of power in order to update and maintain the infrastructure?
8. Localization Time - How much time does it take to determine location from start to finish?

These criteria are used to estimate the performance of each system in theory which will determine which system is prototyped.

B. Comparison of Possible Solutions

Initial research has led to the development of four possible solutions to this engineering problem. Each of these solutions can be evaluated by Liu's criteria, but a full prototype system would be necessary for more than theoretical evaluations.

The first two solutions represent low-level systems which would require custom developed location determination algorithms. As such, the criteria evaluations are heavily dependent on the efficiency of the developed algorithm. The third solution is an open-source software package, and the fourth solution is a proprietary on-the-market product. These solutions are shown in tables 3.1 through 3.4.

Linksys AC1900 Dual Band Open Source WiFi Wireless Router

Accuracy	Dependent on developed algorithm - four antennas per router would most likely give more accurate results
Precision	Dependent on developed algorithm
Coverage	Approximately 30 meters from each beacon
Update Interval	Dependent on developed algorithm
Computational Cost	Dependent on developed algorithm and update interval
Infrastructure	Requires extensive infrastructure overhaul - dependent on size of building/warehouse, but each router cost \$250
Offline Computing	Variable
Localization Time	Dependent on developed algorithm

Table 3.1 - Linksys Open Source WiFi Router Solution Evaluation

USB WiFi Dongle with Linux Computer

Accuracy	Dependent on developed algorithm - most likely least accurate of all solutions
Precision	Dependent on developed algorithm
Coverage	Approximately 30 meters away from each beacon
Update Interval	Dependent on developed algorithm
Computational Cost	Dependent on developed algorithm and update interval. Most likely most power-hungry of all solutions due to separate Linux box
Infrastructure	Fairly costly. Each beacon consists of a USB WiFi dongle and Linux box (may be able to lower cost using Raspberry Pi, etc.)
Offline Computing	Variable
Localization Time	Dependent on developed algorithm

Table 3.2 - USB WiFi Dongle Solution Evaluation

SubPos (Subterranean Positioning System)

Accuracy	Developer website claims within 1 m ² accuracy depending on hardware and number of nodes
Precision	Fairly consistent
Coverage	Approximately 30 meters from each beacon (WiFi standard)
Update Interval	Can be adjusted to preference
Computational Cost	Dependent on update interval
Infrastructure	Requires additional infrastructure. Each beacon would cost approximately \$3 when purchased in bulk
Offline Computing	Variable - beacon locations may need to be explicitly programmed
Localization Time	< 2 seconds

Table 3.3 - Subterranean Positioning System Solution Evaluation

Cisco Hyperlocation [2]

Accuracy	Given as between 1 and 3 meters with extra antenna module
Precision	Fairly precise
Coverage	Recommended 1 AP for every 2500 ft ²
Update Interval	Variable. Cisco gives refresh rate anywhere from 1 to 10 seconds
Computational Cost	Dependent on update interval but tied to current WiFi usage
Infrastructure	Requires minimal overhaul if currently using a Cisco system, but components are expensive and software also expensive
Offline Computing	Physical locations of each AP must be mapped in the software
Localization Time	< 1 second - associated with current WiFi usage

Table 3.4 - Cisco Hyperlocation Solution Evaluation

After theoretical evaluations of the four solutions, the Subterranean Positioning (SubPos) system [3] is determined to be the best solution to focus the scope of this project. The SubPos system is a software package licensed under GNU General Public License which can be easily modified, tested, and installed on inexpensive hardware. This solution is chosen for the following reasons:

1. **Inexpensive to test and quick to implement** - the SubPos software is free under GPL and hardware components would be inexpensive to buy and test - especially compared with Cisco's Hyperlocation at a cost of \$560 per node.
2. **Claimed accuracy** - several tests of the SubPos system have claimed accuracy to within one meter or better. This is the desired accuracy of the final system.
3. **3D location determination** - the system is built right away to handle location determination in 3-space rather than simply 2 dimensions.
4. **Easily modifiable** - since the SubPos software is open source under GPL, the system can be modified as necessary to try and match any remaining project goals.

The SubPos system is currently written as a client-side software package. Our initial plan if the client-side tests are successful is to take advantage of the open-source property of the software. It can be modified to be server-side to fit the project needs.

IV. EVALUATION OF THE SUBPOS SYSTEM

To evaluate the SubPos system, ESP8266 Wireless Modules are used as access points. Encoded in their SSID is information such as their latitude, longitude, altitude (if using 3D), transmission power, and device ID.

Upon request, an Android phone running the SubPos application searches for all SSIDs in the area that represent the SubPos nodes. The phone decodes the SSID to get the node's position and uses the node's signal strength to locate itself in the area. Several tests were run in an outdoor environment with a location errors of anywhere between 0.5 meters and 2 meters. An example of the test results is shown in Figure 4.1.

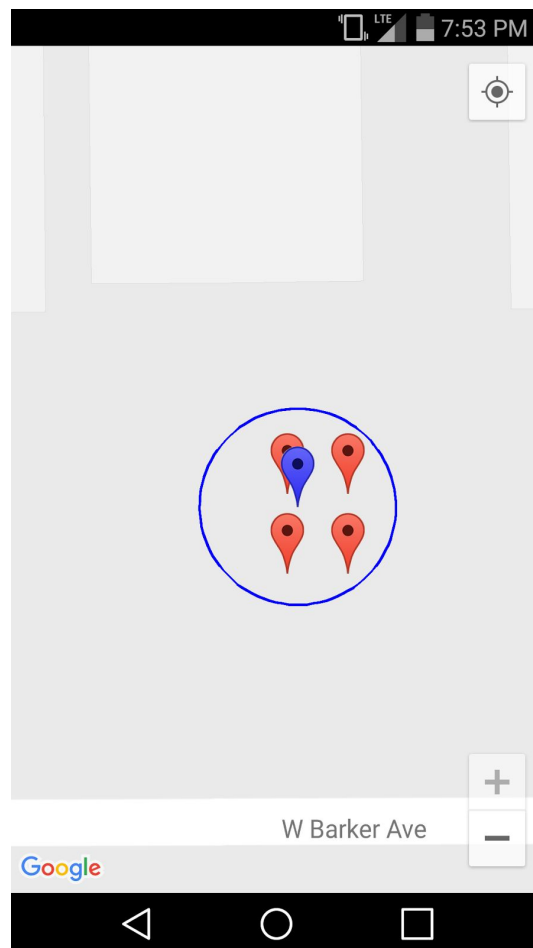


Figure 4.1 - Location determination test results

In Figure 4.1, the red markers along the outside of the square represent the SubPos nodes which are acting as access points. The blue marker represents the application's estimation of the phone's position. The blue circle shows the uncertainty of the estimation.

The accuracy of the SubPos tests show that it may be possible to utilize the full system on a production level. Unfortunately, the problem still remains that the SubPos system is a client-side package. In the context of Panduit's problem, this means that sensors would be determining their own location. The project desires a central server calculating sensor positions with as little influence from sensors as possible.

While examining the source code of the SubPos system, it is determined that it would be too complicated to convert the system directly to be server-side. The conversions are possible, but in order to maintain the desired amount of control and customization over the system, it would be better to recreate a similar system from scratch.

Overall, the SubPos system represents a fruitful start to this project. The system demonstrates the fundamentals of wireless location determination and provides ideas for how to improve accuracy and usability. This project's custom system is developed in three stages: distance determination, localization, and creation of a server.

V. DETERMINATION OF A CUSTOM DISTANCE DETERMINATION EQUATION

The first step in wireless location determination is calculating the distance between nodes whose physical locations are known and the sensors whose physical locations are not. In order to perfect location determination as a whole, distance determination must be improved. Finding this universal, accurate method of distance determination using wireless signals is not a new problem [7]. Distance calculations can be affected by transmitting device, receiving device, weather, environment, interference, and more.

The SubPos system utilizes the following equation to calculate distance:

$$s = 10^{\frac{tx - rx - 10 \log \frac{4\pi}{cf}}{20\mu}}$$

Where s is the distance from node to sensor, tx is the transmission power, rx is the received signal strength, c is the speed of light, f is the transmission frequency, and μ is the path-loss coefficient. The path-loss coefficient helps deal with the issues of path loss [4] and multipath interference [5].

In order to determine the best method of distance determination, correlation tests were run to compare the received signal strength between sensor and node to the distance between them. Two ESP8266 Wireless Modules were used for the test. One module representing the sensor stayed in place while a second module representing the node was placed at various distances between 1 and 10 meters. The node sent a signal to the sensor which recorded the signal strength of this communication. For the first test, five measurements were taken in one meter increments on a straight line in one direction. Figure 5.1 shows the setup for this test.

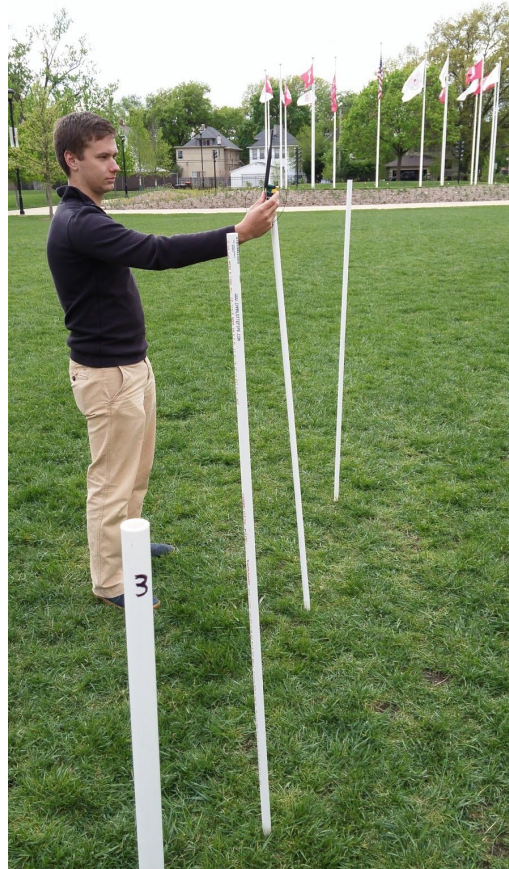


Figure 5.1 - First test setup of distance-strength correlation tests

Figure 5.2 shows the graph of average signal strength vs distance for this first test. The results show that from 1 to 4 meters of distance, the received signal strength drops fairly regularly. After 4 meters, the signal strength hovers just above or just below approximately -71 dBm. While not shown on the graph, distances between 10 and 50 meters also yielded signal strengths of around -71 dBm.

Looking at the datasheet for the ESP8266 modules [6], one of the main technical specifications is “RX Sensitivity”. This represents the point with ESP8266 modules where signal strengths less than the RX sensitivity are capped at the RX sensitivity. For our purposes, the RX sensitivity of our modules is listed as -72 dBm. This means that any signal received weaker than -72 dBm will appear as though it is being received at -72 dBm. Figure 5.2 shows this flat sensitivity cap starting around 5-6 meters.

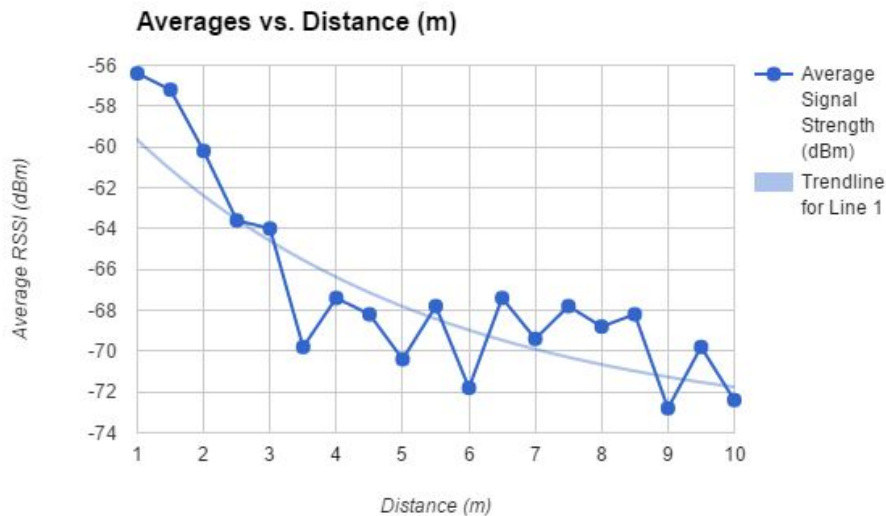


Figure 5.2 - Signal-strength and distance correlation. First test.

While the first test showed some concrete results, our team determined that more data points were needed to further cleanup the graph. In addition, data in the first test was taken in the same direction in a straight line. For a second test, the sensor was placed and a node was rotated around the sensor attached to a string. The string ensures that the node is always the same distance from the sensor as it rotates around getting data from all angles and heights. 30 data points were gathered for each distance and the median was computed. Figure 5.3 shows a graph of the results.

Figure 5.3’s results show the same drop in signal strength between 1 and 5 meters. Aside from a small blip in the data at 6 meters, possibly due to interference or some sort of resonance, data points after 5 meters flatten out around -65 dBm.

After our testing, we began to plug in data points to the SubPos distance equation to see if it is the best option for our system. Substituting the received signal strength for rx and filling in the other variables ($tx = 19.5$, $c = 299.792458$, $f = 2412$, $\mu=2$), we can calculate the distance given by the equation and compare it to the experimental distance values. This error averaged around 20% meaning that the equation used in the SubPos system

provided a distance estimation which differed from the true experimental values by anywhere from 0.5 meters to 4 meters. These comparisons can be found in table 5.1.

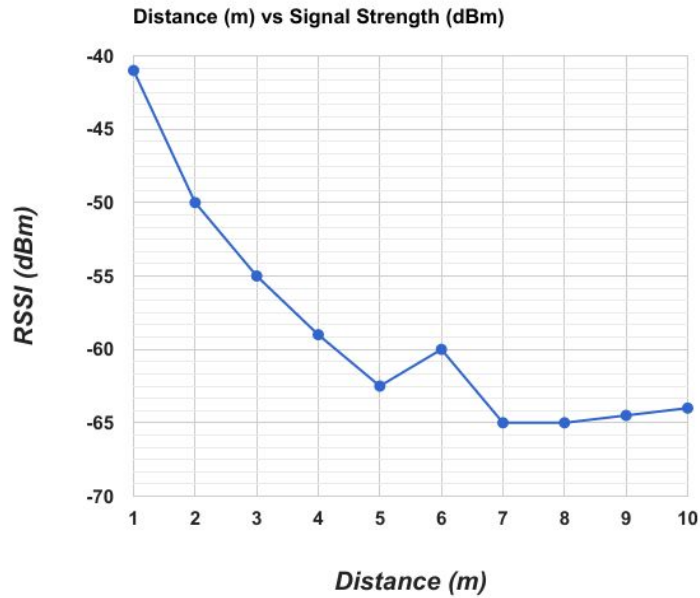


Figure 5.3 - Second test of signal strength and distance correlation

Signal Strength (dBm)	Actual Distance (m)	Calculated Distance (m)	Error (%)
-54.2	0.5	1.77	71.75141243
-56.4	1	2	50
-57.2	1.5	2.1	28.57142857
-60.2	2	2.5	20
-63.6	2.5	3.05	18.03278689
-64	3	3.12	3.846153846
-69.8	3.5	4.36	19.72477064
-67.4	4	3.8	5.263157895
-68.2	4.5	3.97	13.35012594
-70.4	5	4.51	10.86474501
-67.8	5.5	3.88	41.75257732
-71.8	6	4.9	22.44897959
-67.4	6.5	3.8	71.05263158
-69.4	7	4.26	64.31924883

-67.8	7.5	3.88	93.29896907
--------------	------------	-------------	--------------------

Table 5.1 - Comparison of distance calculation using SubPos equation and true distance

Due to the fairly high error of these comparisons, it was decided to run the distance estimation in a different way. Looking at Figure 5.3, the fairly regular progression of the data implies that a statistical regression could be used to determine a best-fit line for the data. Using this trendline, a signal strength could be input to give an estimated distance.

The graph itself most closely represents that of a natural logarithm. In addition, previous research on path-loss indicates that as electromagnetic waves propagate through space, the reduction in power follows a roughly logarithmic decay [9]. Running a logarithmic regression on the data points in Figure 5.3, the following equation is determined:

$$rx = -42.849 + (-10.693) * \ln(d)$$

Where d is the estimated distance and rx is the signal strength. The logarithmic regression gives a coefficient of determination of $r^2=0.94$ showing a fairly strong fit. However, this equation is backwards for the project's purposes as signal strength is known but distance is not. Rearranging the equation yields:

$$d = e^{\frac{rx+42.849}{-10.693}}$$

Where d and rx continue to represent estimated distance and signal strength respectively.

There is another point that must be taken into account when using an equation to determine distance. That is, to make sure when utilizing the equation that a proper domain is ensured. For this equation, signal strength will be given as a negative integer. It is also important to remember that the rx sensitivity of the ESP8266 modules is given as -72 dBm. In Figure 5.3, this rx capping is shown earlier at around -65 dBm. This means that while inputting a signal strength of -72 dBm into the above equation will yield a distance estimation of 15.3 meters, there is no way of determining what the true distance is. In order to avoid this issue, we will only trust signal strength measurements greater than or equal to -65 dBm. Only these measurements will be input to the distance estimation equation.

VI. TRILATERATION / LOCALIZATION

The next step is to calculate the position of the sensor. Now that a distance can be associated with between a node and the desired sensor, the distance measurements can be used to pinpoint

the position of the sensor. First impressions of the problem drive to solve the problem either geometrically or numerically. Both approaches were looked at to determine which method would best match the application. It is desirable for this calculation to be done as quickly as possible to keep the system real time and not bogged down in the calculation.

For the geometric approach, a custom flowchart is needed to fit the application based around a core geometric property. At this stage, it is known that a vector exists between any node and the sensor, where the tail of the vector is on the node and the head of the vector is on the sensor. Thus to solve for the desired vector, the node needs to be set as a reference point for the calculations. To point the vectors inward toward the sensor, the average value of the nodes' position in each dimension is used. To remove the dimensions from messing with the trigonometric functions, the other node must be rotated around the reference node, snapping to the x axis. Using the positions and the distance measurements of two nodes, a triangle can be drawn. From this triangle, the law of cosines can be to solve for the angle of the reference node. Next the angle can solve for the the x and y dimensions of the vector in this "snapped x axis" state. To get the real world vector, the "snapped x axis" vector is simply rotated back the same angle that was used to snap to the x axis about the reference node.

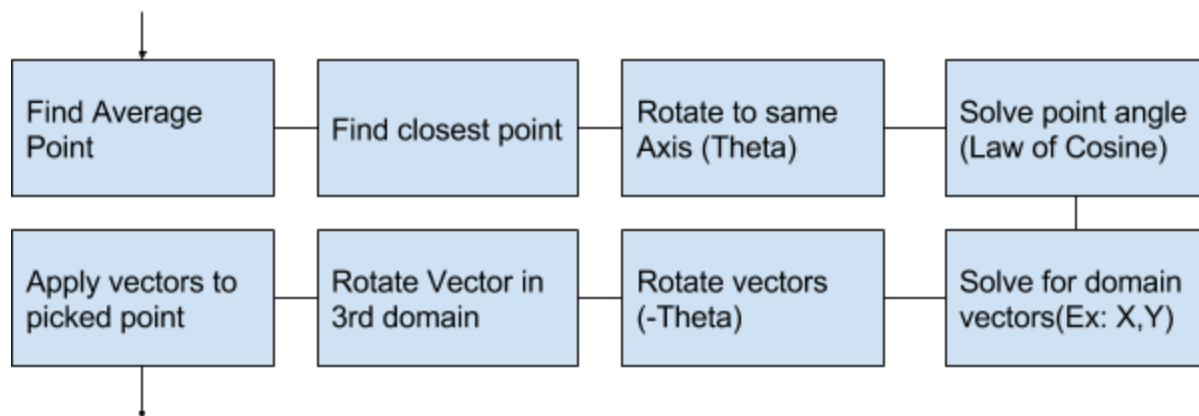


Figure 6.1 Developed Flowchart for Geometric Method

After developing this process, it was found to be too difficult to understand compared to other methods and required the distance measurement to be more accurate than the previously achieved results. In the worst case, the vectors would be applied to each of the nodes and a polygonal area/volume would be left where that the calculation has narrowed the sensor down to. A massive amount of development would have been needed to further progress the approach. Thus, the numerical methods needed to be developed.

After first impressions trying to use a numerical method, it was clear that you must find a solution in a defined amount of iterations, otherwise, the calculation may take too long upon the

need for a fast real-time system. These requirements from the system specifications and from lessons from the geometric method rules out simply using circle equations and solving where they intersect.

For the numerical approach, the Weighted Gradient method proves to be a better route to find the position. In investigating how the numerical problem was solved traditionally in GPS, a vital paper was stumbled upon that compared various methods of solving the localization problem and found that the Nonlinear Least Squared estimation method was the most accurate[8]. With the developments of linearizing the distance measurement in the previous section, this allows other methods to be used. The best method to fit this application from researching different methods is Weighted Gradient method. Why this method is used over other methods such as Least Squared will be discussed shortly. The method weighs the distance measurements against each other by the following equation where $N = \#$ of nodes in range:

$$W \equiv \frac{\sum (DistanceMeasurements) - Measurement_i}{(N - 1) * \sum DistanceMeasurements}$$

After this is calculated the weights are then optimized using Barnes-Fletcher-Goldfarb-Shanno optimization algorithm to calculate the fixed point where our sensor is located. The below figure is an example simulation of this method results. This simulation shows how the optimized gradients are applied from a given reference point to find the position of the sensor.

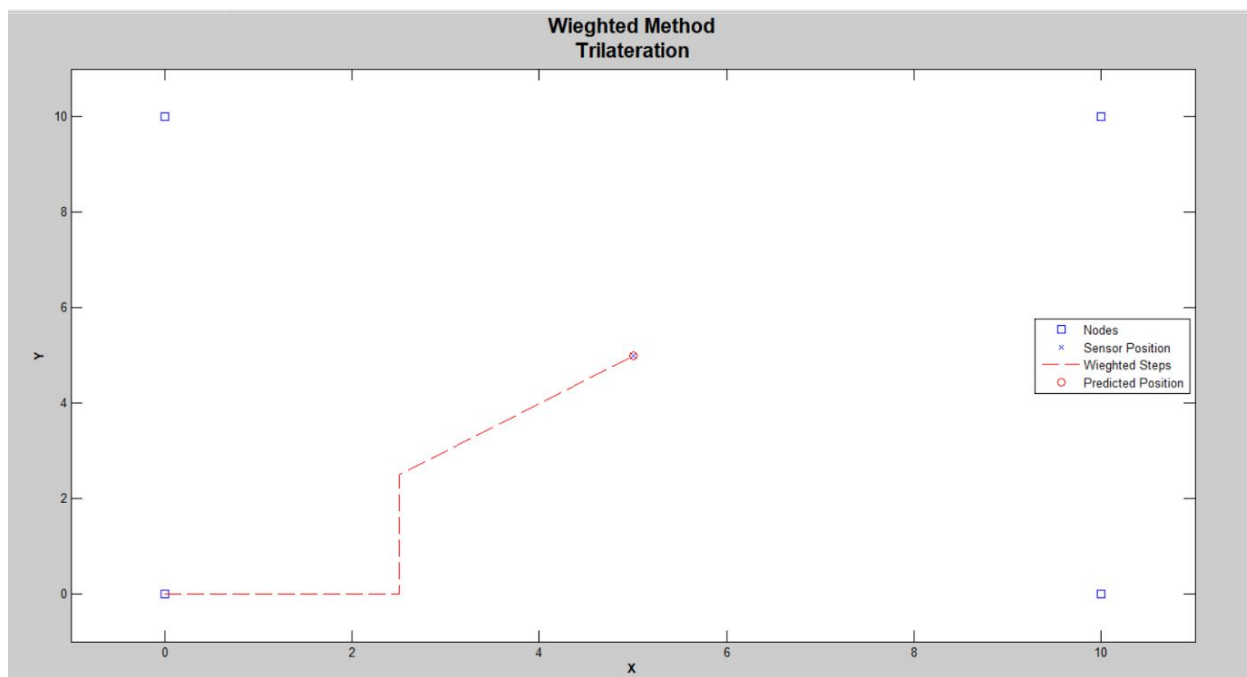


Figure 6.2 Simulation of Weighted Method

Expanding the scope of the decision of what method to pick, the calculation needs to account for measurements skewed from multipath propagation[5]. Our geometric method can not be used because it relies on a perfect distance measurement thus the method would have to be reworked. The Weighted Gradient Method can be accurate while having a noisy distance measurement. This method also focuses on the closest measurement and the longer measurements will have less of an impact on the calculation. Combine with adding a threshold condition below -65 dBm found from our distance measurement research, the impact of multipath effects on our system will be dampened while using the Weighted Gradient method and will maintain decent accuracy. The developed Geometric Method and the Least Squared Method[8] have no answer to take on the multipath problem and will be significantly more skewed. While looking for a quick way to implement the Gradient Method, a python library can be found for the Weighted Gradient method. This proves that this method was heavily trusted. Using the library allowed extremely quick proof of the method and development to move onto server development.

VII. DEVELOPMENT OF A SERVER

To remain battery efficient, a localization method needs to be passive for the sensor. Before, all localization was done on the client side, requiring lots of processing power and active searching. To counter this issue, moving to a server sided process is required.

Using a server, all calculations to locate the sensor can be centralized, rather than the sensors themselves. Instead, the sensors will now only need to ping their WiFi Access Point, which can be controlled by a specific time interval decided by the user. With hundreds of sensors, they would clutter the network listing of WiFi access points on consumer devices such as phones or laptops. In our modules firmware exists the ability to create a hidden network that is unseen by normal devices. This allows our nodes to be able to identify these sensor networks without cluttering the WiFi space for average consumer devices. Being a separate module that is attached to a sensor, this module only needs to be awake to ping its location, then fall asleep. Only using processing power to ping at intervals of only a couple seconds, battery life will be extended for long periods of time.

After a sensor broadcasts a WiFi signal, various nodes in the area need to pick up this signal. These nodes will be installed in an area, with their specific x, y, z location hard-coded into a database. These nodes will require a custom installation dependant to the area it is in. In a large area, nodes need to have a density of at least one for every 5m^3 . In addition, our system does not currently take path-loss directly into account. Multiple nodes will be required in areas with large, signal reflecting surfaces to try and counter this effect. With our system, a greater node density

improves accuracy by compensating for errors between various nodes. This error compensation is done in our trilateration function.

Each node is actively searching for any broadcast signals and isolating signals that do not belong to sensors. When a signal is found, an RSSI strength is calculated on the node, and then the node sends a UDP packet to our server containing the MAC address of our node, the MAC address of the sensor, and the sensor's RSSI.

The server will have an open UDP socket that listens for all packets incoming from our nodes. Once a packet is received the server will attach a timestamp to this packet, and then look in a database for the MAC addresses in the packet. The database contains the hard-coded x, y, and z locations of the node, found based on each node's MAC Address in the database, and the name of the sensor. Using the information, an array is built to include all the node locations and RSSI strengths for each unique sensor. Duplicate node information for each sensor will be removed as well as nodes who have timestamps greater than the sensors unique ping time. New node information will then be inserted.

After the server handles an incoming packet, it will check if a sensor's array contains information for at least three (for 2D localization) or four (for 3D localization) nodes. If requirements are met, the server uses our trilateration function to determine the sensor's location.

Location of the sensors are kept in a database in the case that the server reboots, a sensor can no longer be located or if other errors may occur so that the sensors' last known location can be accessible.

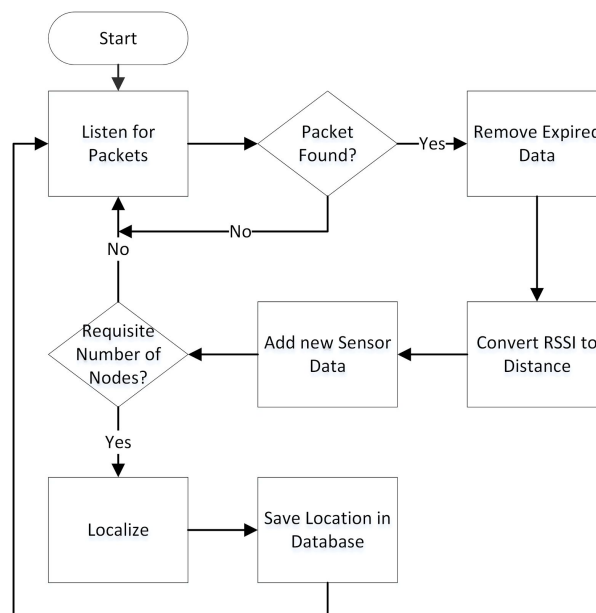


Fig 7.1: Server Functionality Flowchart

VIII. CONCLUSIONS AND FUTURE WORK

After all development was completed, this wireless location determination system had the following specifications and results:

1. Utilizes IEEE 802.11n WiFi running at 2.4 GHz
2. Price of deployment per node of approximately \$5.00. This provides 1 m³ of coverage for about \$0.01.
3. Location determination capability in 2 and 3 dimensions.
4. Sensor need not perform any tasks. At this time, a small module that is in the same physical location of the sensor is responsible for broadcasting SSID. This could later be incorporated into future sensors' firmware.
5. Accuracy in open environment testing of between 10 cm and 1.4 m.
6. Changes in location can be detected after approximately 2 seconds. This delay allows for previous broadcasts to bounce around and decay so as to not interfere with calculations.
7. Database backup for the server implemented using SQLite.

What makes the developed system unique compared to other wireless localization methods is the combination of price and accuracy. While commercial systems exist which claim they can location to within 1 meter, these systems are often very expensive such as the Cisco Hyperlocation system. The developed system provides very similar accuracy in the tested environments with a price of only \$0.01 for 1 m³ of coverage.

The consequences of developing an accurate, inexpensive, and passive wireless location determination system could impact the areas of human safety as well as product distribution. If the system were implemented on any wireless-capable device such as a cell-phone, workers in the field could be monitored without any work on their part. For example, if the field worker does not return by the end of the day, the system could provide a last-known-location to start a search. Large product distribution warehouses, such as those run by Amazon, would also benefit from this system. Attaching a "sensor" to a pallet with rare items would allow the items to be constantly monitored. Should one of the rare items be needed for the first time in several years, there would be no questions as to where the pallet of items is located.

There are several directions to progress in improving this system. With this network covering indoor environments and GPS covering outdoor environments, there may be some value to integrating the system with both environments together. This would allow positioning coverage in any environment. Another direction would be increasing the support for the sensors and nodes. This additional support would develop ideas such as remote updating and sensor & node health.

These developments would allow the nodes to be remain mounted, still allow its software to be updated, and be able to analyze the condition of the sensors and nodes.

The biggest direction for this system is making the nodes into a mesh network. This would make the system more commercially viable. This network is created by node to node communication, passing signal strength measurements and what nodes made those measurements to the server. The server would need a major face lift. It would need filter through the received info for each detected sensor but still do the positioning calculation. To make the passing of information easier a message protocol would have to be developed. The server may also need to store routing tables to increase the overall speed of the network and calculate the most efficient route of sending data back to the server. If the the sensors reading are also passed back to the server, it may also be able to 3D map the measurement of the sensors. For passing of the sensor's measurement, only one node would collect this data and pass it through the network to the server. This collection would need a timestamp to keep the system real time. The nodes would have to make a minor change of being to switch off between being an access point and a broadcast point to pass the data throughout the network. To make the network adaptive to environment needs, it would be advantageous for a node to be able to self-locate its own position. This advancement would allow nodes to be outside the range of the server and the network to significantly cover more area than the current finished developments. An example of the mesh network can be seen in the diagram below.

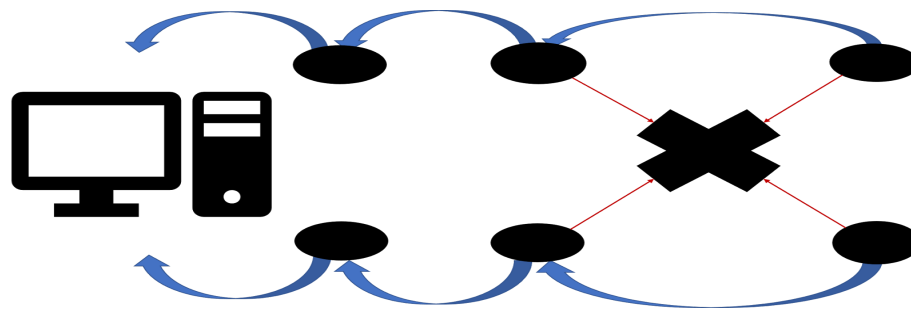


Figure 8.1 Example of Mesh Network flow of data

To further improve the current system a more accurate distance determination can be implemented. Currently the system only uses RSSI strength to determine distance, a method which can be greatly affected in non-open environments. Other methods, using the same system implementation can include: Bluetooth Beacon networks, channel state information (CSI) to isolate bounced signals, and beamforming on wireless AC networks to help determine the direction where a signal is strongest.

These methods could be implemented into the server, while the sensor modules remain untouched and only replacing the node modules to use the hardware requirements needed for the newly used method as well as sending the appropriate data to the servers UDP socket.

References

- [1] Liu, Junjie. "Survey of Wireless Based Indoor Localization Technologies." Washington University in Saint Louis., 30 April 2014. Web. 17 Nov. 2016.
- [2] "Cisco Hyperlocation Module with Advanced Security Data Sheet." Cisco. Cisco Systems, Inc., 12 Sept. 2016. Web. 17 Nov. 2016.
- [3] <http://subpos.org/>
- [4] https://en.wikipedia.org/wiki/Path_loss
- [5] https://en.wikipedia.org/wiki/Multipath_interference
- [6] http://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf
- [7] <http://web.stanford.edu/~skatti/pubs/sigcomm15-spotfi.pdf>
- [8] Hereman, Willy, and William S. Murphy, Jr. *Determination of a Position in Three Dimensions Using Trilateration and Approximate Distances*. Department of Mathematical and Computer Sciences, Colorado School of Mines, 17 Sept. 1995. Web.
- [9] https://en.wikipedia.org/wiki/Log-distance_path_loss_model

APPENDIX A - PARTS LIST

Quantity	Part	Part Cost	Total Part Cost
16	3.3V, 500mA Step-Down Voltage Regulator D24V5F3	\$ 4.25	\$ 68.00
16	HiLetgo ESP8266 Serial WIFI Wireless Module ESP-07 Wireless Module	\$ 6.99	\$ 111.84
3	Highfine 2 x 6dBi 2.4GHz 5GHz Dual Band WiFi RP-SMA Antenna	\$ 9.69	\$ 29.07
2	3.3 V Regulator (5)	\$ 5.71	\$ 11.42
2	9 volt battery clip (5)	\$ 5.19	\$ 10.38
1	USB to UART cable	\$ 5.99	\$ 5.99
1	Header pins (40pinsx25pc), Warning! 2mm, not 2.54mm	\$ 8.08	\$ 8.08
3	Elegoo 6PCS 170 tie-points Mini Breadboard kit	\$ 6.49	\$ 19.47
2	8 Count - Duracell MN1604 9V Volt 6LR61 Duralock Coppertop Alkaline Batteries	\$ 12.99	\$ 25.98
		Total Dev Cost	\$ 290.23

APPENDIX B - SERVER SOURCE CODE

Full source code for all aspects of this project can be found on GitHub at <https://github.com/stanczyk4/ESP-Localization>. The following pages are the raw source for each file which makes up the server.

database.py - responsible for all database access functions

```
import os.path
import sqlite3
import sys

class Database:
    def __init__(self, filepath):
        if os.path.exists(filepath):
            pass
        else:
            print("Database is Missing...")
            sys.exit()
        self.conn = sqlite3.connect(filepath)
        self.c = self.conn.cursor()

    def searchIDfromMACinSensor(self, MAC):
        self.c.execute('SELECT ID FROM sensor WHERE MAC=?', (MAC,))
        return self.c.fetchone()[0]

    def closeDatabase(self):
        self.conn.close()
```

distance.py - responsible for converting signal strength measurements to distances

```
import math

def calcDistance(RSSI):
    return math.pow(math.e, (int(RSSI) + 42.849) / -10.693)
```

main.py - file run to start the server, coordinates all other tasks

```
import atexit
import time
import sys

#local libraries
import database
import server
import packet
import sensorClass
import plot

databaseFilePath = 'database.db'
serverIP = "192.168.2.1"
serverPORT = 5000
plotGraph = True

def programExit(db,serv,graph):
    graph.closeGraph()
    db.closeDatabase()
    serv.closeSocket()
    sys.exit()

def main():
    #create database object db to handle all database functionality
    db = database.Database(databaseFilePath)
    #create server object serv to handle all server operations
    serv = server.Server(serverIP, serverPORT)

    #create plot figure if plotGraph = True, otherwise set myGraph to False
    myGraph = plot.Plot(plotGraph)

    #register an OnExit function to close the database and server in case the program exits
    atexit.register(programExit, db, serv, myGraph)

    #start listening to packets in the background
    serv.listenForPackets()
    sensorData = sensorClass.SensorList()

    while True:
        #checks if we have a packet, if so, do something else loop again
        if not serv.queue.empty():
            newPacket = packet.Packet(serv.queue.get())
            newPacket.sortPacket()

            sensorData.getIndex(newPacket.MAC_Sensor)
            #Now index contains the correct sensor object
```

```

# If we dont have a sensor data object for this sensor
# Create a new one and append to the sensorData list
sensorData.wasFound(newPacket.MAC_Sensor, db)

# Clear out old data from beacons
print "Cleaning expired data...\n"
sensorData.removeOldSensors()

# Add the new values
print "Adding new data values...\n"
sensorData.addNewSensor(newPacket, db.c)

print sensorData.listSensorItems()

# Check if there are 3 or more beacons for this sensorData, if so, Trilaterate it
if sensorData.sensorListLength() > 2:
    loc = sensorData.sensorTrilat()
    print "LOCATION:",loc
    if myGraph:
        myGraph.plotGraph(sensorData.listSensorItems(), loc)

if __name__ == "__main__":
    main()

```

packet.py - a class which defines the structure of the system's UDP packets

```
class Packet():  
    def __init__(self, packet):  
        self.packet = packet  
  
    def sortPacket(self):  
        self.splitData = self.packet.split(',')  
        self.MAC_Sensor = self.splitData[0]  
        self.MAC_Beacon = self.splitData[1]  
        self.Rssi = self.splitData[2]
```

plot.py - responsible for real-time graphing of nodes and sensors

```
import matplotlib.pyplot as plt

class Plot():
    def __init__(self, makePlot=False):
        if makePlot == True:
            #self.plt = plt
            plt.ion()
            self.makeGraph = True
            self.fig = plt.figure()
            self.ax = self.fig.add_subplot(111)
            self.ax.cla()
            self.ax.set_title('Node Localization')
            self.ax.set_xlabel('X Distance (in meters)')
            self.ax.set_ylabel('Y Distance (in meters)')
        else:
            #self.makeGraph = False
            return False

    def closeGraph(self):
        if self.fig:
            self.fig.close()

    def plotGraph(self, nodes, loc):
        self.ax.cla() #clears axis
        #self.ax.clf() #clears figure
        minX = min(x[1][2] for x in nodes)
        minY = min(y[1][3] for y in nodes)
        maxX = max(x[1][2] for x in nodes)
        maxY = max(y[1][3] for y in nodes)
        axisOffset = max(rssi[1][0] for rssi in nodes)

        plt.axis([minX - axisOffset, maxX + axisOffset, minY - axisOffset, maxY + axisOffset])
        for circle in nodes:
            x = circle[1][2]
            y = circle[1][3]
            rad = circle[1][0]
            name = "Node"

            if (circle[0]):
                name = circle[0]

            self.ax.annotate(name, xy=(x,y+.20),horizontalalignment='center',size=8)
            self.ax.add_patch(plt.Circle((x,y),radius=.05, color='k', fill=True))
            self.ax.add_patch(plt.Circle((x,y),radius=rad, color='k', fill=False))

        self.ax.add_patch(plt.Circle((loc.x, loc.y),radius=.06, color='b', fill=True))
        self.fig.canvas.draw()
        plt.show()
```

sensorClass.py - class which defines all methods used for each sensor detected

```
import time

#local library
import trilateration
import distance

class SensorList():
    def __init__(self):
        self.list = []
        self.sensorIndex = 0
        self.found = False

    def getIndex(self, MAC):
        for i in range(len(self.list)):
            if self.list[i].mac == MAC:
                #this is the sensor data object we want
                self.sensorIndex = i
                self.found = True
                break

    def wasFound(self, MAC, db):
        if not(self.found):
            sensorID = db.searchIDfromMACinSensor(MAC)
            newSensor = SensorData(MAC, sensorID)
            self.list.append(newSensor)
            self.sensorIndex = len(self.list) - 1

    def removeOldSensors(self):
        self.list[self.sensorIndex].clearOld()

    def addNewSensor(self, packet, db):
        self.list[self.sensorIndex].replaceData(packet.MAC_Beacon, packet.Rssi, db)

    def listSensorItems(self):
        return self.list[self.sensorIndex].data_dict.items()

    def sensorListLength(self):
        return len(self.list[self.sensorIndex].data_dict.items())

    def sensorTrilater(self):
        return trilateration.trilateration2D(self.list[self.sensorIndex].data_dict)

class SensorData(object):
    def __init__(self, MAC, ID):
        self.mac = MAC
        self.name = ID
        self.data_dict = {}
```

```

def findCombo(self,MAC_Beacon):
    ret = 0
    for beacon in self.data_dict.items():
        # beacon = (mac, [rssi,timestamp])
        if beacon[0] == MAC_Beacon:
            ret = 1
            break
    return ret

def replaceData(self,MAC,RSSI,db):
    # If we are replacing data, we should already know if the combo exists
    # or not
    if (RSSI< -65):
        return
    db.execute('SELECT X FROM node WHERE MAC=?',(MAC,))
    X = db.fetchone()[0]
    db.execute('SELECT Y FROM node WHERE MAC=?',(MAC,))
    Y = db.fetchone()[0]
    db.execute('SELECT Z FROM node WHERE MAC=?',(MAC,))
    Z = db.fetchone()[0]

    est_dist = distance.calcDistance(RSSI)

    self.data_dict[MAC] = [est_dist,time.time(),X,Y,Z]

def clearOld(self):
    for beacon in self.data_dict.items():
        if time.time() - beacon[1][1] > 2:
            del self.data_dict[beacon[0]]

def __str__(self):
    st = str(self.mac)+","+str(self.name)+"\n\n"
    for i in self.data_dict:
        st = st+i+"\n"

    return st

```


server.py - responsible for starting the UDP communication

```
import socket
import threading
from Queue import Queue
import sys

class Server:
    def __init__(self, IP, PORT):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((IP, PORT))
        self.queue = Queue(maxsize=0)

    def closeSocket(self):
        self.sock.close()

    def listenForPackets(self):
        def listen():
            while True:
                data, addr = self.sock.recvfrom(1024)
                self.queue.put(data)

        self.listen_UDP = threading.Thread(target=listen)
        self.listen_UDP.daemon = True
        self.listen_UDP.start()
```

trilat.py - responsible for running the trilateration/localization function

```
import localization1 as lx
def trilat2D(nodes):
    mySolver = "LSE"
    P = lx.Project(mode='2D', solver=mySolver)

    for anchors in nodes.items():
        #add_anchor(ID, (x,y))
        P.add_anchor(anchors[0], (anchors[1][2], anchors[1][3]))

    t, label = P.add_target()
    for anchors in nodes.items():
        #add_measure(ID, distance)
        t.add_measure(anchors[0], anchors[1][0])

    P.solve()
    center = t.loc

    return center
```

¹ <https://pypi.python.org/pypi/Localization/0.1.4>

APPENDIX C - CLIENT SOURCE CODE

Sensor Module Code

```
#include <ESP8266WiFi.h>

const char* ssid = "PSR_SENSOR1";
const char* password = "DONTVENTRY";

boolean wifiError = false;
void setup(){
    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, HIGH);
    boolean result = WiFi.softAP(ssid, password);

    if(result == true){
        digitalWrite(LED_BUILTIN, LOW);
    }else{
        wifiError = true;
    }
}
void loop(){
    if (wifiError){
        digitalWrite(LED_BUILTIN, LOW);
        delay(500);
        digitalWrite(LED_BUILTIN, HIGH);
        delay(500);
    }
}
```

Node Code

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

const char* ssid = "PSR Server";
const char* password = "Panduit1";

WiFiUDP Udp;
const uint16_t portUDP = 5000;

bool scanReady = false;
long lastScanMillis = 0;
long scanInterval = 1500; // interval in time, milliseconds

char myBuffer[40];

void restartConnection(){
    WiFi.disconnect();
    connectToWiFi();
}

void connectToWiFi(){
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED){
        delay(250);
        digitalWrite(LED_BUILTIN, LOW);
        Serial.print(".");
        delay(250);
        digitalWrite(LED_BUILTIN, HIGH);
    }
    Serial.println("Connected");
    digitalWrite(LED_BUILTIN, HIGH);
    scanReady = true;
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println("Boot ON");
    pinMode(LED_BUILTIN, OUTPUT);
    connectToWiFi();
}

void loop() {
    if(WiFi.status() == WL_CONNECTED){
        if(strcmp(WiFi.gatewayIP().toString().c_str(), "0.0.0.0") == 0){
            restartConnection();
        }
    }
}
```

```

//scanNetworks is asynchronous, so don't run multiple sessions
if(scanReady){
    WiFi.scanNetworks(true);
    //WiFi.scanNetworks(true, true); //use this to look for hidden networks
    scanReady = false;
}

int n = WiFi.scanComplete();
//if scan is complete, then n will contain a number greater than 0
if(n >= 0){
    for (int i=0; i<n; i++){
        //Search each network found, if one is a sensor, send data
        if(WiFi.SSID(i).startsWith("PSR_SENSOR")){
            //WiFi.macAddress() returns string of ESP MAC
            //WiFi.RSSI(i) is the RSSI of the found network
            //WiFi.BSSID(i) is the MAC of the found network

            //send info through UDP
            //WiFi.gatewayIP().toString().c_str()
            Udp.beginPacket(WiFi.gatewayIP().toString().c_str(), portUDP);
            //order is in MAC_Sensor,MAC_Beacon,RSSI
            String myPacket = WiFi.BSSIDstr(i);
            myPacket = myPacket + "," + WiFi.macAddress();
            myPacket = myPacket + "," + String(WiFi.RSSI(i));
            myPacket.toCharArray(myBuffer, 40); //mac address = 17bytes? RSSI = 3
bytes? so 17+17+3 = 37, 40 just in case
            Serial.print("Sending packet: ");
            Serial.print(myBuffer);
            Serial.print(" to IP address: ");
            Serial.println(WiFi.gatewayIP().toString().c_str());
            Udp.write(myBuffer);
            //Udp.write(myPacket, length(myPacket)); //Udp.write(buffer,
size) where buffer = outgoing message, size = size of the buffer
            Udp.endPacket();
        }
    }
    WiFi.scanDelete(); //remove the scan from memory
    //make sure at least a set interval has lasted between scans, as we've noticed
    //from testing, multiple scans before a new broadcast signal is sent is
repetitive data
    while((millis() - lastScanMillis) < scanInterval){}
    lastScanMillis = millis();
    scanReady = true;
}
}else{
    connectToWiFi();
}
}

```